TYNE & WEAR

ATARI 8-BIT

USER GROUP

- ☆ Large Selection of
- ☆ Printshop Icons
- ☆ Public Domain Library

Issue 19

January/February 1996

TWAUG NEWSLETTER

EDITORIAL

The management.

John Matthewson

Max Gerum

NOTICE

We regret to inform you that we need to increase the subscription fee, from your next renewel date.

We have tried to keep the fee low as long as we could, but our paper increased double in price and the postage rate is also going up.

The new subscription rate is as follows:

HOME	1 COPY	£2.50
- DO -	6 COPIES	£12.50
EUROPE	1 COPY	£2.50
- DO -	6 COPIES	£13.50
ELSWHERE	1 COPY	£3.50
- DO -	6 COPIES	£16.00

We are very sorry about this increase.

Next issue will be ready by mid-March.

ISSUE CONTENT

DON'T LET BASIC BUG YOU

Part 2

By Mike Bibby

The BASIC Tutorial for beginners.

Before we start, let me give you a warning. The computer will do exactly as you tell it but only what you tell it. It's a very literal machine and in this respect is like my daughter on a mischievious day:

When asked to put on her pyjamas for bed she did exactly as she was told. Of course, I hadn't asked her to take her other clothes off first, had I? You can imagine the results...

Similar things happen with the computer. Say we want the computer to calculate 2+2. Not only do we want it to do the sum but we want it to tell us the answer when it's done it.

We instruct the Atari to write things on the screen with the Basic word **PRINT**. This is a relic from the days when the computer's output, as it is called, was actually printed out on paper rather than on the screen as it is now.

So, to see the answer to 2+2, type:

PRINT 2+2 [Return]

Note that you don't need the = sign as you do on a calculator. [Return] takes care of that. Before continuing try a few simple additions.

Just as the computer does not allow you do use O for 0, so it does not permit to use x for multiply. The computer uses the symbol * instead.

For example try:

PRINT 4*3 [Return]

Minus (-) is straightforward. you'll find it sharing a key with an underline character and a vertical arrow. Divide, however, uses an oblique stroke (/).

For example,

PRINT 12/4 [Return]

Though this may seem at first

DON'T LET BASIC BUG YOU continued

a little odd to you, you have met it when dealing with fractions: 3÷4 is equivalent to the fraction 3/4.

Try:

PRINT 3/4 [Return]

From now on I am going to assume that you accept that before the micro can act on your instructions, they must be sent to it by [Return]. I may therefore omit [Return] from my examples. Make sure that you don't. Before experimenting with further sums of your own devising, I'd like you to try the following sequence:

PRINT 2+8-3

PRINT 4*8/2

PRINT 4*8+2

PRINT 4*(8+2)

If you think carefully about the results you'll see that the computer interprets sequences of sums in the order you learned at school. You do whatever is inside brackets first, then multiplication and division, then finally addition and subtraction.

Now try:

PRINT 2/3

PRINT 10000*10000*10000

PRINT 1/1000

If you have done this correctly, your screen should display:

```
PRINT 2/3
0.666666666

READY
PRINT 10000*10000*10000
1E+12

READY
PRINT 1/1000
1.0E-03
```

The point to stress here is that the computer works to a limit of accuracy. For example, 2/3 is not exactly 0.666666666. The error is well under a

DON'T LET BASIC BUG YOU continued

millionth, though. Still, it must be borne in mind.

Similarly, with especially large or small numbers, the computer saves space by storing them using a scientific notation called exponent format. Here, for example, instead of printing out the answer to 10000*10000*10000 as 1000000000000, it print out the result as 1E+12.

For E, which stands for exponent, you should read "multiplied by 10 to the power of". For example, 1E+12 means "1 multiplied by 10 to the power of 12" which, if your maths is up to it, gives you the correct answer.

If you don't follow all this, don't worry. I've only covered it to warn you about odd looking results to your sums which might pop up and confuse you.

Now let's try to get the computer to print out some words. Let's get it to print out **Hello**. If you cast your mind back to your schooldays (and for some of us that's an awful long throw), you'll remember that when someone says something you surround what that person says with quotation marks (or quotes, for short), such as: He said, "Hello".

In Basic, of course, we don't say words, we PRINT them, but we do surround them by quotes. We omit, however, the comma and full stop.

Try:

PRINT "Hello" [Return]

and the computer should print out **Hello**.

Notice that the quotes are not printed. So to get the Atari Basic to print out a message on its screen we just use PRINT followed by the message surrounded by quotes.

The message inside the quotes is called a string - since the micro considers it to be just a string of letters - or a string literal. The latter is because the

DON'T LET BASIC BUG YOU continued

computer prints out literally, or exactly, what is between the quotes.

So:

PRINT "Hello"

PRINT " Hello"

PRINT " Hello"

give different outputs since in each, different numbers of spaces precede the **Hello.**

Actually, strings do not have to be words. They can be any combination of symbols, including numbers. Just keep them in quotes:

Try the following:

PRINT "4*3"

PRINT 4*3

This should convince you that the computer does print out strings - that is what is between the quotes - literally. When the calculation is in quotes the computer simply echoes the sum on the screen. When the calculation is not in quotes, the computer prints out the answer.

Experiment with printing out various messages on the screen. How long can you make them? Try lower case words as well.

At the moment the computer is responding to our commands as soon as we send them by pressing [Return] but in a calculation or task requiring several steps this can be reather tedious.

It would be more satisfactory to give the computer a whole sequence of instructions that it could get on with rather than spoon-feed it step by step.

This is possible.

Such a sequence of instructions is called a program.

Before we begin to write our own programs, nothing spectacular mind, but enough to give you a quiet glow of satisfaction. Firstly, let's discuss what we did so far.

We learned that to "talk" to the computer we had to speak to it in a language it already

DON'T LET BASIC BUG YOU continued

understood, called Basic. We also learned how to get the Atari to do sums for us and to print out messages, or strings as they are known.

One Basic word we used quite frequently was PRINT, which instructs the micro to write or print things out on the screen. For instance, to do the sum 4+4 we typed:

PRINT 4+4 [Return]

where [Return] means you should press the Return key - this sends the message we have typed to the computer. Hopefully it then responds by printing the correct answer, 8.

Similarly, we could do subtraction, multiplication and division - the symbols for which are -, * and / respectively. Notice particularly the division symbol.

We also learned that to print out messages we had to surround them with quotes, as we do when recording speech. So, to print the message "GOOD MORNING" on the screen we type:

PRINT "GOOD MORNING" [Return]

which causes the message to be written on the screen. Now we can use lower case or small letters, so we can print "Good Morning" by using:

PRINT "Good Morning" [Return]

providing we use our Caps properly.

Notice that PRINT itself remains in capitals. This is because it is a special Basic word - a keyword. For the Atari to realise that it has a special meaning, it must be written in capitals, as must all other Basic words. For the moment, stay in capitals all the time - this will prevent you from falling into this error.

So far we have given the computer one instruction at a time, which it carried out immediately after we pressed Return (assuming we'd typed it

DON'T LET BASIC BUG YOU continued

correctly).

Sometimes, though, we want to give the micro a series of instructions and then tell it to carry them out. For instance, suppose we want the message:

PROGRAMMING

IS

EASY

to appear on the screen. With our step-by-step method, we would have used:

PRINT "PROGRAM-MING"

[Return]

PRINT "IS"

[Return]

PRINT "EASY"

[Return]

But, as you'll see if you try it, this doesn't produce the required effect, since each successive instruction spoils the layout. We need to give the computer the instructions so that it:

1. Prints out PROGRAMMING

2. Prints out IS

3: Prints out EASY

in sequence, without stopping to ask us what to do next. Such a sequence of instructions is called a program. Notice also that the sequence is numbered - after all, the computer needs to know the order in which to carry them out.

Now let's write a program to print out

PROGRAMMING

IS

EASY

We were on the right lines with the first attempt, but this time, let's try numbering our instructions as we enter them.

First of all type:

NEW [Return]

NEW is a Basic keyword that clears out the computer's memory. If you don't do this the program you are typing in might get jumbled up with a previous one - you'll see more clearly how this can happen

DON'T LET BASIC BUG YOU continued

later.

You probably think that you haven't got a program in at the moment, but use NEW anyway, because it is possible that you might have entered a line or two by chance.

Then type:

10 PRINT ''PROGRAM-MING''

[Return]

Notice two things:

- The first instruction is number 10, not number 1. In computing we tend to number our instructions in steps of ten for reasons that will become blindingly obvious later. We call the number of an instruction its line number.
- The computer didn't immediately carry out the instruction - it didn't print out PROGRAMMING after we pressed Return. This is because of the line number. It tells the computer that what follows isn't to be done immediately but is just to be remembered for later as it is just one in a series of instructions. I'll prove that the computer actually does remember it in a moment.

Now type:

20 PRINT ''IS''[Return]

30 PRINT ''EASY''[Return]

What I'm going to ask you to do next should test your faith in me! Clear the screen by typing:

[Shift+Clear]

All your typing should should have disappeared, but don't worry - your work hasn't been wasted. Because of the line numbers, the computer has kept a list of yur instructions in its memory. To see the list, type:

LIST[Return]

and your program should reappear. We'll call it Program 1:

```
10 PRINT "PROGRAM-
MING"
20 PRINT "IS"
30 PRINT "EASY"
```

Program I

An important point coming up now. We have entered a

DON'T LET BASIC BUG YOU continued

program (a numbered sequence of Basic instructions) into the Atari's memory and have got the computer to display those instructions with LIST. We have not, however, told the computer to do these instructions. It's like having written a shopping list - you still have to go to the shops and turn your list into reality.

So to get the computer to actually do, or as we say, run the program in its memory, we type:

RUN[Return]

and, if we've typed it in properly, we should see printed out:

PROGRAMMING
IS
EASY

If you've managed it, congratulations on running your first program. (If not, don't worry, it's probably some simple error. List your program and look for the mistake. You might actually have a message telling you that there is an error in a particular line. What we're about to do next, although it assumes that you have been successful so far, will in fact show you how to correct your mistakes.)

Now let's try to alter our program so that it prints out:

PROGRAMMING
IS
SIMPLE

If you look back at the first program you will see that you need to alter line 30.

Changing line 30 couldn't be simpler - just type in the new line 30, remembering to start with the line number 30, then press Return. The latest version will replace the old version in the computer's memory.

To demonstrate this, type:

30 PRINT "SIMPLE"
[Return]

DON'T LET BASIC BUG YOU continued

and then:

LIST[Return]

You should obtain Program II which is:

```
10 PRINT "PROGRAM-
MING"
20 PRINT "IS"
30 PRINT "SIMPLE"
```

Program II

An examination of this listing should reveal that the new èversion of line 30 has indeed replaced the old one. (Notice also èthat we didn't precede LIST with a line number - we wanted the micro to do it immediately.)

As a final proof that our ammendment has been accepted, type:

RUN[Return]

You should now get the revised message. If you accidentally type line 10 as:

**10 PINT "PROGRAM-
MING"**

then, when you tried to run it you would get an error message.

To rectify such mistakes, simply retype the correct version èof the line 10 and press Return to enter it into the computer.

There are more sophisticated ways of correcting, or editing, a line, but they can wait for a while. For the moment èwe shall simply retype the line, with its line number, and press Return. Of course, if you notice a mistake while you are entering a line, use the Delete key to erase it, then continue typing from that point.

So far I have given you just two programs to run. However, using these models, you can print out virtually any message you want on the screen. Just use line numbers in increments of 10, each line printing out part of the message you want out on the screen, by enclosing it in quotes after PRINT.

An important point about this series is that I'm going to give

DON'T LET BASIC BUG YOU continued

you lots of example programs to type in. Virtually all of them have two things in common:

- They make vital teaching points (otherwise they wouldn't be there in the first place).
- The output - that is, what appears on the screen - is trivial in content and in many cases there are far easier ways of doing it.

Programming is a skill like driving - you can only improve by doing it, not reading about it. Please carry out the examples, however silly or obvious they may seem to you.

Also, and this is far more important, I want you to go beyond the programs - try to alter, adapt and extend them, just to see what happens.

Adopt an experimental approach and a healthy scepticism for my pronouncements. If you are wondering whether something will work, go ahead and try it - you can't hurt the computer from the keyboard, so let your imagination run riot.

You'll learn far more from your own examples than you will by merely echoing mine. And the good thing is that you get such prompt feedback from a computer. If what you write isn't acceptable you'll soon get an error message.

So what I'd like you to do now is to spend a good time writing simple ''message'' programs for the computer to run. For some reason, in my experience in computing classes the messages tend to become quite scurrilous. There's one thing I've never been too sure of - is it slander or libel when it appears on a VDU?

Remember, type NEW before each new program, and use line numbers for each instruction. It's also good policy to LIST your program before you RUN it, just to make sure that all is as you intend.

Now suppose we intend to

DON'T LET BASIC BUG YOU continued

alter Program II so that it printed out the message:

PROGRAMMING

IS

RATHER

SIMPLE

We need a line in there between 20 and 30 to print out RATHER. Well, 25 is a number between 20 and 30, so let's try:

25 PRINT "RATHER"

[Return]

If you list it you'll see that the program has now become Program III:

```
10 PRINT "PROGRAM-
MING"
20 PRINT "IS"
25 PRINT "RATHER"
30 PRINT "SIMPLE"
```

Program III

So line 25 has "crept in" between 20 and 30. Even though we entered it out of order, the Atari stores it in memory in its correct numerical position. Try running the program as final confirmation.

This ability to insert lines into programs is the reason our line numbers go up in steps of 10 when we are writing programs - it leaves us plenty of spare line numbers in between for when we are patching them up.

Now enter Program IV:

```
10 PRINT CHR$(125)
20 PRINT "ATARI"
30 PRINT "USER"
```

Program IV

remembering to press Return after typing each line.

Now LIST it. Is there a phantom line 25 in there? If so, you didn't type NEW after the last program - the lines 10, 20 and 30 of the latest program have replaced those lines in the old program. But as the new program doesn't have a line 25, the old one remains to ruin your program. The moral is to

DON'T LET BASIC BUG YOU continued

use NEW before entering a new program. If you have got an unwanted line 25, don't worry - you can easily get rid of it by typing:

25[Return]

This will delete the line since you replace the old line 25 with a new line which contains nothing - which the computer then "forgets". This method holds good for deleting any line from a program - simply type out the line number, then press Return.

I'd better explain what line 10 does: It clears the screen. I don't want to devote much space to it here, so let's just accept it for the moment - we'll explain it fully later in the series.

You'll soon see that it works when you run the program.

Now let's try to print out our message with blank lines between. We can use a line containing just PRINT to obtain a blank line, so Program V should do the trick:

```
10 PRINT "CHR$(125)"
15 PRINT
20 PRINT "ATARI"
25 PRINT
30 PRINT "USER"
```

Program V

Now try Program VI:

```
10 PRINT "CHR$(125)"
20 PRINT "HELLO";
30 PRINT "OUT";
40 PRINT "THERE"
```

Program VI

The output you will get is:

HELLOOUTTHERE

That is, each successive string is printed after the preceding one. The semicolon stops the next string being printed on a new line, "gluing" it to the end of the previous string printed.

Notice that since there are no spaces inside the strings, none appear between the words

DON'T LET BASIC BUG YOU continued

when they are printed out together.

Try to get the message to appear legibly by rewriting the program with appropriate spaces in the strings. Also notice that you can obtain the same output, far more simply, with Program VII:

```
10 PRINT "CHR$(125)"
20 PRINT "HELLO OUT
THERE"
```

Program VII

However, as I said above, the programs I present to you are for making teaching points, which does not necessarily imply showing you the most efficient methods.

Experiment with joining up the output of successive PRINT statements with the use of the semicolon until you feel confident about it.

And now for something completely different.

Try running Program VIII. I think the effect is pretty impressive.

So far all our programs have merely copied back onto the screen what you have typed in. This program shows how, with the addition of one line (line 60), you can obtain a huge increase in the amount of output. It is this ability, to repeat a simple operation rapidly, that gives the Atari much of its power.

```
10 PRINT "I"
20 PRINT "FEEL"
30 PRINT "DIZZY"
40 PRINT
50 PRINT
60 GOTO 10
```

Program VIII

If things are happening a little too fast for you, you can temporarily suspend action by pressing:

[Control] + 1

This freezes the action until you press:

[Control] + 1

DON'T LET BASIC BUG YOU continued

once more.

What is happening is that the computer follows lines 10, 20, and 30 and prints out:

I (line 10)

FEEL (line 20)

DIZZY (line 30)

followed by two blank lines. It then encouters line 60, which tells it to go back to line 10. It duly does so and prints out:

I (line 10)

FEEL (line 20)

and so on, until it reaches line 60, when it goes back to line 10 and so on ad infinitum. Notice that when the screen is full, it scrolls up to make more room.

Now the name for such a condition in a program, where you keep on repeating lines of code (as the program lines are known), is a loop.

We say here that we are in an unconditional loop because we haven't given the program any conditions for it to cease repeating itself. This is bad programming practice - compulsively introspective computers are not useful machines!

To stop such unconditional loops you have to interrupt them from "outside" by pressing the Break key. As you'll see, you get a message telling you which line the program stopped at.

If you want to have some fun with an unconditional loop, try Program IX. It repeatedly prints out an arrow composed of asterisks such as:

```
   *
  ***
 *****
*******
  ***
  ***
  ***
  ***
```

which will scroll upwards off the screen.

DON'T LET BASIC BUG YOU continued

```
10 PRINT "    *"
20 PRINT "   ***"
30 PRINT "  *****"
40 PRINT " *******"
50 PRINT "   ***"
60 PRINT "   ***"
70 PRINT "   ***"
80 PRINT "   ***"
90 PRINT "   ***"
100 PRINT
110 PRINT
120 PRINT
130 PRINT
140 GOTO 10
```

Program IX

always naughty, can you see what else is going wrong with Program X?

- In next issue we'll discover new ways to create programs.

```
10 PRINT "CHR$(125)"
20 PRINT "THIS IS"
30 PRINT "VERY SILLY"
```

Program X

130 XE COMPATIBLE 256K UPGRADE FOR THE 600XL

By Jeff Popp

I remember a while back seeing a survey that showed the percentage of the 8-Bit Atari market that each model held. The 600 XL held a lowly 2%. Ok, it was a low end machine, made for the folks that weren't able to drop the bucks to get something more powerful. Some mail order houses are now offering 600's for $29.99, and many units are being sold. Should thease Atarians remain locked into a 16K, TV hook-up only world? No, I say! They should be able to hack and torture their computers just like

256K UPGRADE FOR THE 600XL continued

the rest of us. It was in this light that the 256K upgrade for the 600XL was born.

When this mod is installed in conjunction with the monitor mod, it will make your 600 everything a 130XE is and more, in a smaller package! Kits for both of these mods are available from Best Electronics in San Jose, CA. (408)243-6950.

Claus Bucholz designed the first 256K upgrade for the 800XL and later modified it to be 130XE compatible. This circuit for the 600XL is an adaption of Claus' 800XL mod, and will operate with the ramdisk autoboot program written by him, anyone interested in the internal workings of this mod should get Claus' article where he describes the cicuit in detail. The 800XL uses 64K x 1 memory chips, and the 41256 memory chips used in the upgrade are directly pin compatable. No such luck with the 600XL. However, the 256K chips can be utilized with only minor modifications. The address lines of the 256K chips (A0-A8) do not match up with the address lines of the 16K x 4 chips used in the 600, but this dosen't matter, since the computer won't care where it puts something as long as it can find it there later. On to the meat of the mod...

This is a fairly complicated mod, and good soldering skills are a must. Please do not attempt this mod unless you are experienced in this kind of procedure. Take your time and double check your work as you go, since troubleshooting this thing can be a real pain!

The parts needed for this mod are:

1. 8 ea. 41256-15 memory chips
2. 1 ea. 74LS153 Dual 4 to 1 multiplexer
3. 1 ea. 74LS139 2 to 4 decoder
4. 1 ea. 33 ohm 1/4 watt resistor
5. A foot or 3 of 30 gauge insulated wire.

256K UPGRADE FOR THE 600XL continued

If U9 has a part No of C012296, you will also need:

6. 1 ea. 74LS158 Quad inverting 2 to 1 multiplexer
7. 1 ea. 74LS393 Dual 4-bit counter.

The first step is to open up the 600 and pull the RF sheild. Remove the chips and sockets at U11 and U12. Bend up the following pins so that they do not enter the sockets and hang out in free space.

U5 pin 3

U6 pins 9, 10, 11, 12, 13, 14

U18 pins 8, 9, 10

U21 pins 12, 13, 14, 15, 16

Piggyback 4 of the 41256 chips, soldering all of the pins together except for 2 and 14. Bend these straight out and clip off the thin part as shown in figure 1.

Repeat the process for the second set of 4 chips.

U11 and U12 will now refer to to the spot on the board where the chips are placed, while U11 A, B, C, D and U12 A, B, C, D will refer to the two stacks of chips, A being at the bottom and D at the top. Solder a piece of wire about 2 inches long into each of the following holes:

U11 pins 2, 3, 15, 17

U12 pins 2, 3, 15, 17.

These will be used later.

Now very, very, carefully, solder in place the two stacks of memory chips into U11 and U12 so that pin 8 of the stacks goes into the spot on the circuit board allowed for pin 9 of the origional memory chips, bending up pins 1 and 16 of the stacks so that they do not enter the holes below them.

Jumper U11A-D pin 16 into U11 pin 18.

Do the same with U12A-D.

Now the rough part...

Follow the wiring hook-up shown in the schematic of figure 2.

Got all that done?

256K UPGRADE FOR THE 600XL continued

Take a break! You deserve it!

The Address lines feeding U5 and U6 on the 600 are different than those on the 800XL so we have to make them match up. Carefully cut the traces going to the following pins:

U5 pins 6, 10, 13

U6 pins 3, 6

You're coming down the home stretch!! Don't quit now or you'll have a 600XL door-stop. (No you didn't start with one!)

Jumper between these places on the solder side of the circuit board:

U3 pin 24 to U5 pin 6

U3 pin 21 to U5 pin 13

U2 pin 1 to U5 pin 10

U2 pin 2 to U6 pin 3

U2 pin 3 to U6 pin 6

U18 pin 8 to U18 pin 10

Lastly, on the part side of the board, jumper the bent up pin 3 of U5 to the non-bent up pin 25 of U3. Cover the ram stacks and any exposed connections with strips of electrical tape to avoid shorting to the RF shield.

Hook up the board to your TV (or monitor if you have the monitor upgrade), plug in the power supply, hold your breath, turn on the power switch and run!

Really, don't worry, chances are very small that something will even get warm. See if the unit will boot up. If it does, then run the on-board memory test. You should see three lines of memory test blocks instead of the single line you're used to. Be sure to boot while holding down the option key because with BASIC enabled, you'll only see two and a half lines of blocks.

To fully test the extended memory, you will need to load the ramdisk program, or a 130XE program that uses the extra bank of memory.

Believe it or not, you're done! Go get some sleep!

PRINT-WORKS- FOR ATARI 1029

Reviewed

by Alastair. G. Fraser

If you are sitting with a big colour-printer, or the latest laser-job, this article is not for you. But if you are the average impecunious Atari owner, who has seraped his way up to a disk drive and a 1029 printer, this program opens up new horizons. After years of tolerating the limitations of the 1029, "Print-Works, the Document Processor for the Atari 1029 Printer", gave me some very pleasant surprises. When you load the program, pressing "Option", you come onto the "Editor" screen, ready to start work. The number of characters free is shown, also the commands for cursor-moving, etc. The "Inverse" key opens up a list of over 100 symbols, including the "Pound" sign, which you can put into your document, and which will be printed exactly as you see them. Press "Inverse" again to return to the "Editor" screen. Press "Esc" for the command line, which appears at the bottom of the screen, showing FILE, PRINT, HELP, SETTINGS. The horizontal arrow keys will move the highlight to your desired heading, then "Return". "HELP" accesses the disk to bring up 21 variations to help your layout, and pressing any key takes you to a further screen for working with blocks of text. "SETTINGS" Gives you variations of Fonts,(3) types of script,(Italics, Outline, Shadow,Etc) margins, justification, etc, etc, - arrow keys move the cursor. "PRINT" When highlighted, press "Return", and you see PRINT, SHOW, PREVIEW, PRINTER, which lets you see exactly how your document will look, and scroll 80 cols,. Press "Return" again and your printing! Instructions for other printers(printer driver programs) are included, also instructions for importing text

PRINT-WORKS- FOR ATARI 1029

files from other proccessors, in a comprehensive manual. All in all, I find this program excellent, and have only one slight criticism:- I cannot find a method of formatting disks within the program, so you need to keep a number of formatted disks available. This Excellent program is available from: Micro-Discount 265 Chester Road, Streetley, West Midlands, B74 3EA Tel 0121 3535730 Fax 0121 3521669 Price 4 Pounds 95p

ATARI ASSEMBLER COURSE

By John Demar.

Why Assembly Language

To better understand the need for Assembly Programming, let's compare the different types of programming languages: interpre-ted, compiled, and assembled. The language we are most familiar with, Atari BASIC, falls in the

ATARI ASSEMBLER COURSE

first category. When an interpreted language runs, each command is examined and converted to its machine code equivalent and then executed. Because of this "one-at-a-time" method, these languages tend to be slower but require less memory. Also, the language itself requires memory (8K in the case of Atari BASIC) even with no program loaded into the computer! Some languages, like Forth, have a very fast yet small interpreter. Others like Pascal are larger and slower. The closest a language can come to the speed of machine code is a truly compiled language, such as ACTION!, that generates the "native code" of the computer itself (the 6502 microprocessor in our case). Compiled languages differ from interpreted languages in the way they are translated into machine code. Unlike an interpreter, a com-

ATARI ASSEMBLER COURSE continued

piler generates the equivalent machine code for the complete program before it is run. Although this step takes extra time, it needs only to be done once and the resulting machine-code equivalent program will run much faster every time. Lastly, assembled language is a means of writing programs directly in machine code but we have the use of words to represent the lowest level operations of the microcomputer. The operational codes or "Op-codes are represented as three-letter symbolic words or "mnemonics". Unlike the other types, assembly language allows the programmer to optimize the speed, size, and timing of a program by talking in the computers own words. These advantages will easily justify the longer developement time in many applications, such as games, where timing is intricate, the size must be minimized and, naturally, it has to be fast!

Atari System Architecture

In order to design good software with your Atari Computer, you need a good understanding of the hardware. Because Assembly Language deals directly with the machine itself, you will become good friends with "ANTIC", "GTIA", "POKEY", and the "PIA". These chips are the work-horses of the Atari and help out the 6502 with the various duties of the computer system. We will get into more detail with them through out the course.

For now:

Read: "DE RE ATARI", Chapter 1, (note figure on page 1-3).

6502 CPU Description

ATARI ASSEMBLER COURSE continued

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0

ACCUMULATOR

X INDEX REGISTER

Y INDEX REGISTER

N	V	X	B	D	I	Z	C

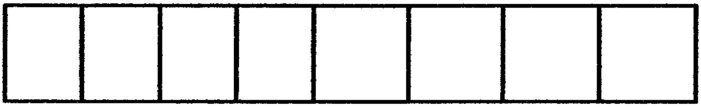
PROCESSOR STATUS REGISTER

PROGRAM COUNTER HIGH

ATARI ASSEMBLER COURSE continued



PROGRAM COUNTER LOW



STACK POINTER

6502 Microprocessor

Internal Registers

Registers - A,X,Y

A "register" is a memory cell, internal to the microprocessor, in which eight "bits" or binary digits are stored. The accumulator is the register where data must be stored to be operated upon. The "X" and "Y" registers are normally used to keep data for indexing into memory or as a counter in a loop.

Processor Status Register.

The next most important register in the 6502 is the process status register. As its name implies, the "P" register holds the status of many conditions in the CPU. Each bit in this register is a flag, either a "1" or a "0", that may be used as a test for branching or looping in a program.

The symbolic letter for each status flag is shown in the figure above within corresponding bit of "P" register. The "C" or "carry" flag is set (equal to one) when an arithmetic operation results in a value greater than 255 (the highest value for 8 bit numbers). The "Z" flag is set when an operation results in a value equal to

ATARI ASSEMBLER COURSE continued

zero. The "I" flag is used during an "interrupting" or time-sharing portion of the program. The "D" flag is set when "decimal" arithmetic is used and reset to "0" for "binary" math. The "B" flag is set when the 6502 Break command has occurred. The "X" flag is not used.

The "V" flag is set when an operation results in a value greater than +127 or less than -128 and is used for signed arithmetic. The "N" flag, or negative flag, is equal to the seventh bit of a result; this is also used for signed arithmetic. Each of the status bits may be changed in a program except for the break flag.

Program Counter - PC.

The program counter holds the value of the memory location where the next machine code instruction is located. It has a "high" and "low" byte so that 64K bytes of memory can be "addressed" with the 16 bits. Also, the "PC" is changed by the CPU as a result of a branch, jump, or subroutine in the program so that the next operation is found. We will see some other uses of the program counter later on.

Stack Pointer - S

The Stack is another very useful aspect of the 6502 CPU. It is located in memory on "Page-One" (hex 0100 to 01FF) and is reserved for the the 6502 and the programmer for limited usage. The stack begins at the end of page-one (hex 01FF) and is filled on a last-in, first-out basis. The Stack Pointer holds the location of the "top" of the stack.

Review of Binary and Hex Math.

Because we are dealing directly with the microprocessor itself, it is very important to understand the operations of binary numbers and their hexidecimal equivalents. We can review this briefly during the

ATARI ASSEMBLER COURSE continued

session if there are questions. But, get a good understanding before we begin. A very good tutorial appears in the "ANALOG" magazines in the column "Boot Camp" by Tom Hudson. If you can get hold of copies of this American magazine, you will find them a mine of information.

SYNFILE+

(KNOWLEDGE IS NEVER OLD, ITS ALWAYS CURRENT. *This old article is worth reading).*

There are actually four different files that together make up one SynFile+ datafile. The files can be identified by their extenders. They are .TBL, .CNF, .IDX, and .Dxx. The. TBL file contains a description of the database form. The .CNF file contains the number of records and disks. The .IDX file is the current index data. And, the .Dxx files are the actual data files.

SYNFILE + cont.

THE .TBL FILE

The .TBL file is actually made up of three tables and two numeric entities. The three tables are the Definitions table, the Name table and the Special Data table. Each table is headed by a two byte length entry, and then the data bytes. If the table has a zero length, there is only a length entry.

THE DEFINITIONS TABLE

The first table, the Definitions table, consists of a sequence of eleven bytefield definitions. There is one entry for each field in the database. The format of each entry is:

Byte # Use

0 screen column position

1 screen row position

2-3 field name offset
and name length

4 field type byte

5 field data length

SYNFILE + continued

6 field mask length

7 field special offset

(low 8 bits)

8 field data offset

(low 8 bits)

9 field data offset

(high 3 bits)

field special offset

(high 5 bits)

10 field decimal position

The first two bytes simply give the row and column position of the start of the field name. The screen is 40 columns by 21 lines and 0,0 is the upper left hand corner.

The field name is actually stored in the second data table. The low 11 bits of this 16 bit entry are the offset to the field name in the Name table. The high 5 bits are the field name length.

The low four bits of the field type entry, identify the field type. The high bit of the field type is a flag for the justification of the field. If it is set, the field is right justified. There are eleven field types in SynFile+ and each has a number associated with it. They are:

ID number Field Type

0 ASCII field

1 floating point

2 cumulative computed

3 table look-up

4 dollar

5 record number

6 date

7 integer

8 counter

9 conditional

10 computed

The field data length is the length of the data stored in the disk record. For ASCII fields, it is one greater than the field mask length. For floating point, cumulative, dollar and computed fields, it is 6 bytes. For table look-up and conditional fields, it is one byte. All other fields have a data length of 2

SYNFILE + continued

bytes.

The field mask length is the length of the mask (underlines) for the characters respectively.

The special offset is 13 bits split between byte 7 and 9. Byte 7 has the low 8 bits of data, and the Special Data table has the high 5. It is used by computed, cumulative, conditional and table look-up fields. The counter field also uses this entry, but not as a pointer. Counter fields use it as the increment for the field.

The field offset is 11 bits long. The low 8 bits are stored in byte 8, and the high 3 bits are stored in the low 3 bits of byte 9. This is the offset from the start of the record to the start of the data from this field.

The field decimal position is just that. It tells SynFile how to display floating point numbers. The current version of SynFile only uses the low 4 bits of this byte.

The others are reserved for future use. If the value of this byte is 15, the field will be displayed in floating point, which is - however - the Atari FP ROM formats the number. For any other value, SynFile will force the display of a decimal point and N digits to the right of the decimal.

THE NAME TABLE

The Name table contains all the field names. Each field name is stored as a text entry, with no delimiters or separators between entries. The names may NOT be stored in the same order as the fields entries in the Definitions table(this may occur if the form is edited in the CREATE module of SynFile).

THE SPECIAL DATA TABLE

The Special Data table contains all formulas and table look-up field datas.

FORMULAS

Formulas for computed,

SYNFILE + continued

cumulative, and conditional fields are stored as a sequence of command tokens. The CREATE module parses the user entered formula, and converts it to a tokenized RPN formula. As retrieved, all commands use the top 1 or 2 entries from the stack. If the high bit of a token is set, then the field referenced is an integer(16 bit data) field, a FLOAT will automatically be executed on the field when the data is retrieved. The command tokens are:

TOKEN VALUE
COMMAND

0 + (add the top two values)
1 - (subtract the top value from second value)
2 *
3 /
4 LOG (take the natural LOG of top number)
5 LOG10 (take the common log of top number)
6 EXP
7 EXP10
8 ABS
9 SORT
20 numeric constant
30 = (set true flag if top 2 entries are equal)
31 >
32 <
33 <>
34 >=
35 <=
126 current data
127 END (end of formula flag)

Numeric constants(20) are stored in 6 byte internal floating point representations of the number. Entries 30 to 35 are used to compare 2 numbers on the stack for use in conditional operations.

To interpret a computed field, use the special data offset to set a pointer to the formula. The retrieve tokens one byte at

SYNFILE + continued

a time until the END token is found. The value on the top of the stack at that time is the result of the calculation.

TABLE LOOK-UP

Table Look-up entries are stored in packed entries(the lenght of the entries is equal to the length of the longest entry, not to the mask length). The field data for a table look-up field is the table entry number. The first byte of a look-up table is the number of entries in the table, and the second byte is the length of the entries in the table. Each entry is stored as a text string (with no length byte) and is right justified in the table with underline characters (ASCII 95) as filler on the left.

CONDITIONALS

Conditional entries are a combination of formula entries and table look-up entries. The special offset for a 2 entry look-up table. Immediately after the look-up table data is the formula data for the conditional field. The last thing in the .TBL file are two numeric entries. Each entry is 16 bits long(2 bytes, stored low byte, high byte). The first entry is the field count, the second entry is the total record length in bytes. The record length is one greater than the sum of the field data lengths.

THE .CNF FILE

The .CNF file contains eight entries. Each entry is 16 bits long(2 bytes, stored low byte, high byte). The entries are:

1 The length of the index array entries

2 the total number of records in the datafile

3 the number of data disks in the datafile

4 the number of active records in the datafile

5 the current record number

6 the number of index fields

7 the current value of the

SYNFILE + continued

counter field

8 the sort direction

(ascending - descending flag)

THE .IDX FILE

The length of the .IDX file varies with the total number of records in the file, and the length of the current index. (Both these numbers are in the .CNF file). The first 32 bytes of the .IDX file are the current index field. There are 16 two-byte entries, the first byte of each entry is the field number, the second is the length of the index data for that index entry. Any unusued entries have undefined values, usually 0.

The T.W.A.U.G.

Management

wishes all

Subscribers a very

HAPPY 1996

FOR SALE

Various Atari hardware including:

Upgraded 130XE,

1050 Disk Drives with various upgrades, i.e.:

U.S. Doubler, Happy, Laser, Quad Board.

Other items also available including Touch Tablet and Light Gun

Ring John Tel 0191 2626897.

Atari XMM801 Printer in perfect working order. Reason for selling, surplus to requirements. Price 40 pounds or nearest offer.

Contact: G. Lewsley,

Tel 01482 856179

Armstrad PCW 9512 Personal Computer Word Processor including Printer, in perfect working order.

For more information -

Ring Max Tel 0191-5866795

LACE

The London Atari Computer Enthusiasts

As a member of LACE you will receive a monthly newsletter and have access to a monthly meeting. They also support the **ST** with a good selection of PD software.

The membership fee is

£8.00 annually

Write to:

Mr Roger Lacey

41 Henryson Road,

Crofton Park, London,

SE4 1HL

O.H.A.U.G.

The OL'HACKERS ATARI USER GROUP INC.

O.H.A.U.G. is an all 8-bit user group in the STATE of NEW YORK U.S.A.

They are producing a bi-monthly disk based newsletter. The disk comes with its own printing utility, which lets you read the content of the disk and also print the content in columns.

They have a large **PD Library**

Contact: **Mr. A.Pignato**

O.H.A.U.G.

3376 Ocean Harbor Drive

Oceanside, N.Y. 11572, USA

DISK CONTENT

This issue disk and progams has been submitted by **Robert De Letter.**

The content of side A:

There are four games on this side of the disk.

1 - Climber, use joystick to move the climber. Press fire button to lift his legs when those spiders get close, touch these insects and you loose a life. Easy to follows.

2 - Unicom, a ball game, use joystick to move racket.

3 - Whipping Top, use joystick to move top and press fire button to stop Top.

4 - Expedition, guide plane though maze, easy to follow.

5 - Joystick Type, instruction on screen.

Also demo's for music and colours.

Side B: There are four COM files:

1 - Cuttlemania 2 - Into Deep

3 - Submission and Nadral, had no time to check them, don't know how they play.

MICRO-DISCOUNT

Offers
The complete Mail Order
service for Atari 8 Bit
XL/XE users



Contact Micro-Discount
265 Chester Road.
Streetly
West Midlands.
B74 3EA.
England

Tel
0121 353 5730

Fax
0121 352 1669